Middle East Technical University

Department of Computer Engineering

CENG 111

Fall 2018

Take Home Exam 3 [ver. 1.1]

REGULATIONS

Due date: 27 December 2018, Thursday, 23:59 *(Not subject to postpone)*

Submission: Electronically. You should save your program source code as a text file named `the3.py` and submit it to us via the course's COW page.

Team: There is **no** teaming up. This is an EXAM.

Cheating: Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

INTRODUCTION

In this THE, you will be doing some physics and have the chance to visualize your solutions graphically. Your task is to implement a simulation of a gravitational many-body problem (see the figure on the next page).

In this problem, you will be dealing with *particle*s that have *mass* (denoted by m), which is an invariant property. In addition to mass, the particles have dynamic properties, characterized by two vectors: namely the position $\vec{r}$, and the velocity $\vec{v}$. In this THE, for the sake of simplicity, all vectors (position and velocity) will be two-dimensional.

To predict (i.e., compute) a particle system's behavior in time, we have to know the masses and the initial values of the dynamic properties (i.e., positions and velocities) of the particles. Then, the laws of physics allow us to predict the future of the dynamics[1].

Our system will consist of n particles. We will label our particles with integer subscripts. For example, m_4 will denote the mass of the 4^{th} particle. The values in the following table will be provided as input.

Particle	Mass	Initial position	Initial velocity
p_1	m_1	$(x0_1, y0_1)$	$(v0x_1, v0y_1)$
p_2	m_2	$(x0_2, y0_2)$	$(v0x_2, v0y_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
p_n	m_n	$(x0_n, y0_n)$	$(v0x_n, v0y_n)$

In the figure on the next page, you see an example of a many-body particle system with 3 particles.

[1] We confine ourself to classical mechanics, which is quite correct at the macro scale. If we go down to atomic sizes, then this approach will no more predict correct results and you must pick a quantum mechanical approach.

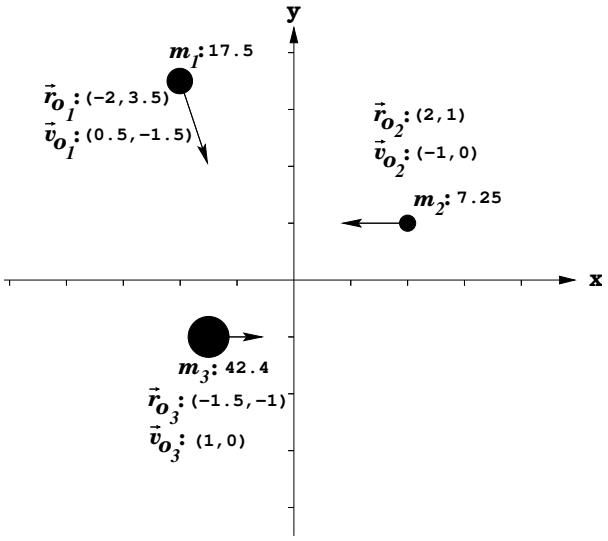
Updating the system

Consider a single particle: As you know, velocity means "*change in position in unit time*". Normally, we express this fact as a derivative, considering "unit-time" as an infinitely small number:

$$\vec{v} = \frac{d\vec{r}}{dt}$$

If we discretize this (i.e., approximate the infinitely small dt by some 'small' number), then we write the above derivative as:

$$\vec{v} = \frac{\Delta\vec{r}}{\Delta t}$$



The difference now is that Δt is just a 'small' number. Then, at any time t, if a particle is at position $\vec{r}_t$, then after the *unit-time* update, the new position $\vec{r}_{t+\Delta t}$ will be:

$$\vec{r}_{t+\Delta t} = \vec{r}_t + \Delta t \cdot \vec{v} \tag{1}$$

Equation 1 always holds, even if the velocity is changing.

If the particle is under any force, then its velocity will change. So, we should also update the velocity $\vec{v}$ after *unit-time* has passed. The change in the velocity is called *acceleration* and is denoted by the vector $\vec{a}$. Newton's second law provides a relation that gives us the *acceleration* at any time the particle will attain, if we know the force acting on the particle at time t as well as its mass:

$$\vec{a} = \frac{\vec{F}}{m}$$

By definition, $\vec{a}$ is:

$$\vec{a} = \frac{d\vec{v}}{dt}$$

Again, if we discretize the time and consider a 'small' *unit-time*, a velocity $\vec{v}$ will be updated according to:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \Delta t \cdot \frac{\vec{F}_t}{m} \tag{2}$$

In order to calculate all the dynamics of a single particle, we have to know the forces acting on it. Force is *superposable*, which means that if there are two forces $\vec{F}_1$ and $\vec{F}_2$ acting on a particle, you can assume that these forces can be substituted by a single force $\vec{F}_{sum}$, which equals

$\vec{F}_1 + \vec{F}_2$. Hence, generally speaking, if a particle is under k number of forces, the resulting force is the vector sum of all of them:

$$\vec{F}_{sum} = \sum_{i=1}^{k} \vec{F}_i$$

Now, what kind of force is exerted by a particle p' on the particle p? In this THE, we will consider a *gravitational* force. As you now, the gravitational force is inverse quadratic proportional to the distance between the two particles (p and p'):

$$\vec{F} = G \cdot \frac{m_p \cdot m_{p'}}{\|\vec{r}_{p'} - \vec{r}_p\|^2} \cdot \hat{u}_{rr'}$$

where G is the so-called *universal gravitational constant* (a real number). Furthermore, $\hat{u}_{rr'}$ is the unit vector in the direction from particle p towards p'. In other words, $\hat{u}_{rr'} = (\vec{r}_{p'} - \vec{r}_p)/\|\vec{r}_{p'} - \vec{r}_p\|$ So, we could have written the force as:

$$\vec{F} = G \cdot \frac{m_p \cdot m_{p'}}{\|\vec{r}_{p'} - \vec{r}_p\|^3} \cdot (\vec{r}_{p'} - \vec{r}_p)$$

If we wish to calculate the resulting force acting on a particle p_i by all other particles making use of the *superposition principle*, we can write:

$$\vec{F}_{i_{sum}} = \sum_{\substack{j=1 \\ j \neq i}}^{m} G \cdot \frac{m_i \cdot m_j}{\|\vec{r_j} - \vec{r_i}\|^3} \cdot (\vec{r_j} - \vec{r_i})$$

Taking all constants out and rearranging the order of subtraction :

$$\vec{F}_{i_{sum}} = G \cdot m_i \cdot \sum_{\substack{j=1 \\ j \neq i}}^{m} \cdot \frac{m_j}{\|\vec{r_j} - \vec{r_i}\|^3} \cdot (\vec{r_j} - \vec{r_i}) \tag{3}$$

With the equations (1), (2) and (3), it is possible to calculate new positions and velocities for all particles after a *unit-time* Δt has passed *(Provided that the initial positions and velocities are given).*

PROBLEM

You will be given the G constant, the unit-time Δt, and the mass and the initial (position and velocity) values for a set of particles as a list. This list will be provided to you by a function call of `get_data()`, which will be provided by the evaluators (i.e., us) at the time of grading.

You are expected to write a function that you will exactly name as `new_move()`, which will take no arguments and (at each call) return the $\Delta\vec{r}$ ($\Delta\vec{r} = \Delta t \cdot \vec{v}$) values for each particle (for the Δt time interval that has passed). This returned list is simply the information of how much each particle will move in the x- and y-directions. The evaluators will be able to observe the whole event simulation by successive calls to `new_move()` .

What you are supposed to submit is the `the3.py` file holding the function `new_move()` (and its helpers). However, we provide you with a visual tool, so that you can visualize how your `new_move()` performs. This Python code, when loaded, attempts to load your Python code that lives in `the3.py`, makes the necessary initializations, opens up an 800×800 screen, places circles of sizes proportional to the masses of the particles and displays the action by internally calling your `new_move()` successively.

HOW-TO-USE `draw.py`

The use of `draw.py` is optional. It serves only visualization purposes. Your code `the3.py` will be evaluated using black-box batch testing and not visually.

- Place `the3.py`, `draw.py`, `evaluator.py` in your working directory and in Python do a

  ```
  from draw import *
  ```

  We provide sample `the3.py` and `evaluator.py` files with dummy `new_move()` and `get_data()` function definitions for your convenience. The sample `new_move()` does no gravitational force calculation at all, and always returns a constant result; it is only for you to see how `draw.py` works.

- `draw.py` uses a coordinate system where the origin is at the top-left corner. x-axis is in the left-to-right direction and y-axis is in the top-to-bottom direction.

- There are two global variables used by `draw.py`. If you want to alter their default values, do so by editing their assignment lines in `draw.py`.

  `DELAY`: The delay (time in milliseconds) between two successive `new_move()` calls. The default is 200 (ms).

  `SCALE`: A floating point constant to determine which screen coordinate corresponds to which physical coordinate. The relation is simple:

  $$\langle \text{Screen coordinate} \rangle = \langle \text{Physical coordinate} \rangle \times \texttt{SCALE}$$

  If you want to have a zoom-out view, you can set `SCALE` < 1.0. The default is set to be 1.0.

- To avoid name clashes, do not re-define something with the names of the global variables and functions of `draw.py`.

SPECIFICATIONS

- There will be at least 1 particle and at most 20 particles.

- To get your input data, you shall execute a call to the function `get_data()` in `the3.py`. This call will return a list like:

  $$[G, \Delta t, [m_1, x0_1, y0_1, v0x_1, v0y_1], [m_2, x0_2, y0_2, v0x_2, v0y_2], \cdots, [m_n, x0_n, y0_n, v0x_n, v0y_n]]$$

  G is the gravitational constant. Δt is the 'small' unit time. The following lists provide the initial values for the particles (namely the mass, initial x and y coordinates, followed by the initial velocity's x- and y-components). All numbers will be floating point values. Multiple calls to `get_data()` in a single run will return the same values.

- In your `the3.py`, you are allowed to import only from the `math` module. You cannot import form any other file/library.

- Evaluating the correctness of the updates calculated by your solution will require performing floating point comparison (by our scripts). We will consider two values f_1 and f_2 the same if $|f_1 - f_2| < 0.001$ is true.

- To test your own code, you will need a `get_data()` function. For that purpose, you should construct your own `get_data()` function. We suggest that you write the code that defines your `get_data()` into a file named `evaluator.py`, which will be automatically loaded by the `draw.py` package.
  - Do not define this `get_data()` (that you will be using for your own testing) in the `the3.py` file.
  - Do not submit your `evaluator.py` file or any other file except `the3.py`. As evaluators, we are not interested in them. We will implement our own `get_data()`.
- Make sure that `new_move()` is implemented in `the3.py`, and conforms to the specifications. You are free to implement helper functions as much as you like. Moreover, you can (and probably will) use global variables. Do not execute a call of `new_move()` in `the3.py` since this is the job of the evaluators.

  Each call to `new_move()` should return a list of the form

  `[[`Δx_1`,`Δy_1`],[`Δx_2`,`Δy_2`], ··· ,[`Δx_n`,`Δy_n`]]`

  where the Δx_i, Δy_i values are the coordinate increments for the i^{th} particle's position in a Δt time (certainly these values can be negative as well as positive).
- Do not perform any error checks. The input will be error free.
- You can define other functions or variables in `the3.py` as long as there are no name clashes with other functions and variables in the other files.
- You can only use the concepts covered in the lectures until the deadline of THE3.

GRADING

- Comply with the specifications. Since your returned results will be evaluated automatically, non-compliant results will be considered as incorrect by our evaluation system.
- Your program will be tested with multiple data (a distinct run for each data).
- Any program that performs only 30% and below will enter a glass-box test (eye inspection by the grader TA). The TA will judge an overall THE2 grade in the range of [0,30]. The glass-box test grade is not open to negotiation nor explanation.
- A program based on randomness will be graded zero.